SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	8/25/97	August 1993 - May 1997

4. TITLE AND SUBTITLE
Algorithms for Processing Large-Scale Data

5. FUNDING NUMBERS
DAAH04-93-G-0076

6. AUTHOR(S)
Jeffrey S. Vitter

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Duke University
Computer Science Departmetn
Box 90129
Durham, NC 27708-0129

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 31513.1-MA-SDI

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The PI and coauthors have developed the first known algorithms for sorting, problems in computational geometry, graph problems, and different forms of range searching that are simultaneously optimal in terms of storage space usage and I/O performance. The PI has worked on how to implement these algorithms in practice using a powerful I/O programming environment called TPIE.

The working group discussed the strategic directions and challenges in the management and use of ---storage systems---those components of computer systems responsible for the storage and retrieval of data. The performance gap between main and secondary memories shows no imminent sign of vanishing, and thus continuing research into storage I/O will be essential to reap the full benefit from the advances occurring in many other areas of computer science. We identified a few strategic research goals and possible thrusts to meet those goals.

14. SUBJECT TERMS

15. NUMBER IF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Final Report
Grant No. DAAH04–93–0076
August 1993 – May 1997
"Algorithms for Processing Large-Scale Data"
Duke University

Jeffrey S. Vitter
Dept. of Computer Science
Duke University
Durham, N.C. 27708–0129

1 Introduction

Today's massively parallel computer systems are capable of performing many important scientific computation and data processing tasks at speeds orders of magnitude faster than the sequential supercomputers they are beginning to replace. Unfortunately, high speed processing is not enough; programs also have to be able to get data into and out of the machine as quickly as the processors are able to process it. The so-called input/output (I/O) bottleneck turns out to be particularly acute when one attempts to use massively parallel systems to manage large spatial databases and to do large-scale processing of information, especially of a geometric nature. This problem has received inadequate prior attention.

Keeping a computer's processing elements supplied with data is the job of the I/O system. The goal of this project work was to develop provably good techniques whereby I/O systems can be effectively integrated into massively parallel computers so that they can be put to work not only in the analysis of data but in its management, storage, and retrieval as well.

2 Summary of project

The PI and coauthors have developed the first known algorithms for sorting, problems in computational geometry, graph problems, and different forms of range searching that are simultaneously optimal in terms of storage space usage and I/O performance. The PI has worked on how to implement these algorithms in practice using a powerful I/O programming environment called TPIE. A more detailed summary of the work appears below.

Recently, in [18], as the working group report on Storage I/O for Large-Scale Computing for the ACM Workshop on Strategic Directions in Computing Research, we discussed the strategic directions and challenges in the management and use of *storage systems*—those components of computer systems responsible for the storage and retrieval of data. The performance gap between main and secondary memories shows no imminent sign of vanishing, and thus continuing research into storage I/O will be essential to reap the full benefit from the advances occurring in many other areas of computer science. We identified a few strategic research goals and possible thrusts to meet those goals.

2.1 Summary of important results

2.1.1 Sorting

In [5], we provided the first optimal algorithms in terms of the number of input/outputs (I/Os) required between internal memory and multiple secondary storage devices for the problems of sorting, FFT, matrix transposition, standard matrix multiplication, and related problems. Our two-level memory model is new and gives a realistic treatment of *parallel block transfer*, in which during a single I/O each of the D secondary storage devices can simultaneously transfer a contiguous block of B records. The model pertains to a large-scale uniprocessor system or parallel multiprocessor system with D disks. In addition, the sorting, FFT, permutation network, and standard matrix multiplication algorithms are typically optimal in terms of the amount of internal processing time. The difficulty in developing optimal algorithms is to cope with the partitioning of memory into D separate physical devices. Our algorithms' performance can be significantly better than those obtained by the well-known but nonoptimal technique of disk striping. Our optimal sorting algorithm is randomized, but practical; the probability of using more than ℓ times the optimal number of I/Os is exponentially small in $\ell(\log \ell) \log(M/B)$, where M is the internal memory size.

In [6], we introduced parallel versions of two hierarchical memory models and give optimal algorithms in these models for sorting, FFT, and matrix multiplication. In our parallel models, there are D memory hierarchies operating simultaneously; communication among the hierarchies takes place at a base memory level. Our optimal sorting algorithm is randomized and is based upon the probabilistic partitioning technique developed in the companion paper for optimal disk sorting in a two-level memory with parallel block transfer. The probability of using ℓ times the optimal running time is exponentially small in $\ell(\log \ell) \log D$.

In [2], we presented an optimal deterministic algorithm called Balance Sort for external sorting on multiple disks. Our algorithm improves upon the randomized optimal algorithm of [5] as well as the (non-optimal) commonly-used technique of disk striping. It also improves upon earlier sorting algorithm in that it has smaller constants hidden in the big oh notation, it is possible to implement using only striped writes (but independent reads), and it has application to parallel memory hierarchies.

In [3], we presented an elegant deterministic load balancing strategy for distribution sort that is applicable to a wide variety of parallel disks and parallel memory hierarchies with both single and parallel processors. The simplest application of the strategy is an optimal deterministic algorithm for external sorting with multiple disks and parallel processors. Our two measures of performance are the number of I/Os and the amount of work done by the CPU(s); our algorithm is simultaneously optimal for both measures. We also showed how to sort deterministically in parallel memory hierarchies. When the processors are interconnected by any sort of a PRAM, our algorithms are optimal for all parallel memory hierarchies; when the interconnection network is a hypercube, our algorithms are either optimal or best-known. The constant factors are very small and the best known, suggesting that the algorithm has definite practical merit.

In [15], we propose a simple, efficient, randomized D-disk mergesort algorithm called SRM that uses a forecast-and-flush approach to overcome the inherent difficulties of simple merging on parallel disks. SRM exhibits a limited use of randomization and also has a useful deterministic version. The upper bound we derive on expected I/O performance of SRM indicates that SRM is provably better than disk-striped mergesort (DSM) for realistic parameter values D, M, and B. Average-case simulations show further improvement on the analytical upper bound. Unlike previously proposed optimal sorting algorithms, SRM outperforms DSM even when the number D of parallel disks is small.

2.1.2 TPIE

In recent years, I/O-efficient algorithms for a wide variety of problems have appeared in the literature. Thus far, however, systems specifically designed to assist programmers in implementing such algorithms have remained scarce. In [19] we describe TPIE which is a system designed to fill this void. It supports I/O-efficient paradigms for problems from a variety of domains, including computational geometry, graph algorithms, and scientific computation. The TPIE interface frees programmers from having to deal not only of explicit read and write calls, but also the complex memory management that must be performed for I/O-efficient computation.

In [12] and [16], we looked at the development of TPIE and describe its use in scientific computation. We discussed algorithmic issues underlying the design and implementation of the relevant components of TPIE and present performance results of programs written to solve a series of benchmark problems using our current TPIE prototype. Some of the benchmarks we present are based on the NAS parallel benchmarks, while others are of our own creation. We demonstrated that the CPU overhead required to manage I/O is small and that even with just a single disk the I/O overhead of I/O-efficient computation ranges from negligible to the same order of magnitude as CPU time. We conjecture that if we use a number of disks in parallel this overhead can be all but eliminated.

2.1.3 Graph problems

In [7] we presented a collection of new techniques for designing and analyzing efficient external-memory algorithms for graph problems and illustrate how these techniques can be applied to a wide variety of specific problems. Our results include:

- *Proximate-neighboring.* We presented a simple method for deriving external-memory lower bounds via reductions from a problem we call the "proximate neighbors" problem. We use this technique to derive non-trivial lower bounds for such problems as list ranking, expression tree evaluation, and connected components.

- *PRAM simulation.* We gave methods for efficiently simulating PRAM computations in external memory, even for some cases in which the PRAM algorithm is not work-optimal. We apply this to derive a number of optimal (and simple) external-memory graph algorithms.

- *Time-forward processing.* We presented a general technique for evaluating circuits (or "circuit-like" computations) in external memory. We also used this in a deterministic list ranking algorithm.

- *Deterministic 3-coloring of a cycle.* We gave several optimal methods for 3-coloring a cycle, which can be used as a subroutine for finding large independent sets for list ranking. Our ideas go beyond a straightforward PRAM simulation, and may be of independent interest.

- *External depth-first search.* We discussed a method for performing depth first search and solving related problems efficiently in external memory. Our technique can be used in conjunction with ideas due to Ullman and Yannakakis in order to solve graph problems involving closed semi-ring computations even when their assumption that vertices fit in main memory does not hold.

Our techniques apply to a number of problems, including list ranking, which we discuss in detail, finding Euler tours, expression-tree evaluation, centroid decomposition of a tree, least-common ancestors, minimum spanning tree verification, connected and biconnected components, minimum

spanning forest, ear decomposition, topological sorting, reachability, graph drawing, and visibility representation.

In [20] we considered the problem of using disk blocks efficiently in searching graphs that are too large to fit in internal memory. Our model allows a vertex to be represented any number of times on the disk in order to take advantage of redundancy. We give matching upper and lower bounds for complete d-ary trees and d-dimensional grid graphs, as well as for classes of general graphs that intuitively speaking have a close to uniform number of neighbors around each vertex.

2.1.4 Computational Geometry

In [21] we gave new techniques for designing efficient algorithms for computational geometry problems that are too large to be solved in internal memory, and we use these techniques to develop optimal and practical algorithms for a number of important large-scale problems in computational geometry. Our algorithms are optimal for a wide range of two-level and hierarchical multilevel memory models, including parallel models. The algorithms are optimal in terms of both I/O cost and internal computation.

Our results are built on four fundamental techniques: distribution sweeping, a generic method for externalizing plane-sweep algorithms; persistent B-trees, for which we have both on-line and off-line methods; batch filtering, a general method for performing K simultaneous external-memory searches in any data structure that can be modeled as a planar layered dag; and external marriage-before-conquest, an external-memory analog of the well-known technique of Kirkpatrick and Seidel. Using these techniques we are able to solve a very large number of problems in computational geometry, including batched range queries, 2-d and 3-d convex hull construction, planar point location, range queries, finding all nearest neighbors for a set of planar points, rectangle intersection/union reporting, computing the visibility of segments from a point, performing ray-shooting queries in constructive solid geometry (CSG) models, as well as several geometric dominance problems.

These results are significant because large-scale problems involving geometric data are ubiquitous in spatial databases, geographic information systems (GIS), constraint logic programming, object oriented databases, statistics, virtual reality systems, and graphics. This work makes a big step, both theoretically and in practice, towards the effective management and manipulation of geometric data in external memory, which is an essential component of these applications.

In [22] we developed efficient new external-memory algorithms for a number of important problems involving line segments in the plane, including trapezoid decomposition, batched planar point location, triangulation, red-blue line segment intersection reporting, and general line segment intersection reporting. In GIS systems, the first three problems are useful for rendering and modeling, and the latter two are frequently used for overlaying maps and extracting information from them. To solve these problems, we combine and modify in novel ways several of the previously known techniques for designing efficient algorithms for external memory. We also develop a powerful new technique that can be regarded as a practical external memory version of fractional cascading. Except for the batched planar point location problem, no algorithms specifically designed for external memory were previously known for these problems. Our algorithms for triangulation and line segment intersection partially answer previously posed open problems, while the batched planar point location algorithm improves on the previously known solution, which applied only to monotone decompositions. Our algorithm for the red-blue line segment intersection problem is provably optimal.

2.1.5 Range Searching and Databases

In [14], we presented a new approach to designing data structures for the important problem of external-memory range searching in two and three dimensions. We based our data structures on the novel concept of B-approximate boundaries, which are manifolds that partition space into regions based on the output size of queries at points within the space. Our data structures answer a longstanding open problem by providing three dimensional results comparable to those provided by Subramanian and Ramaswamy for the two dimensional case, though completely new techniques are used. Ours is the first 3-D range search data structure that simultaneously achieves both a base-B logarithmic search overhead and a fully blocked output component.

In [1], we examined I/O-efficient data structures that provide indexing support for new data models. The database languages of these models include concepts from constraint programming (e.g., relational tuples are generalized to conjunctions of constraints) and from object-oriented programming (e.g., objects are organized in class hierarchies). Let n be the size of the database, c the number of classes, and t the size of the output of a query. Indexing by one attribute in the constraint data model (for a fairly general type of constraints) is equivalent to external dynamic interval management, which is a special case of external dynamic 2-dimensional range searching. We presented a semi-dynamic data structure for this problem which has optimal worst-case space $O(n/B)$ pages and optimal query I/O time $O(\log_B n + t/B)$ and has $O(\log_B n + (\log_B^2 n)/B)$ amortized insert I/O time. If the order of the insertions is random then the expected number of I/O operations needed to perform insertions is reduced to $O(\log_B n)$. Indexing by one attribute and by class name in an object-oriented model, where objects are organized as a forest hierarchy of classes, is also a special case of external dynamic 2-dimensional range searching. Based on this observation we first identify a simple algorithm with good worst-case performance for the class indexing problem. Using the forest structure of the class hierarchy and techniques from the constraint indexing problem, we improve its query I/O time from $O(\log_2 c \log_B n + t/B)$ to $O(\log_B n + t/B + \log_2 B)$.

In [17], we presented a space- and I/O-optimal external-memory data structure for answering stabbing queries on a set of dynamically maintained intervals. Our data structure settles an open problem in databases and I/O algorithms by providing the first optimal external-memory solution to the dynamic interval management problem, which is a special case of 2-dimensional range searching and, as discussed in [1], a central problem for object-oriented and temporal databases and for constraint logic programming. Our data structure simultaneously uses optimal linear space and achieves the optimal output-sensitive I/O query bound and I/O update bound. Our structure is also the first optimal external data structure for a 2-dimensional range searching problem that has worst-case as opposed to amortized update bounds. Part of the data structure uses a novel balancing technique for efficient worst-case manipulation of balanced trees, which is of independent interest.

In [13], for the first time the problem of estimating alphanumeric selectivity is studied in the presence of wildcards. Success of commercial query optimizers and database management systems (object-oriented or relational) depend on accurate cost estimation of various query reorderings. Based on the intuition that the model built by a data compressor on an input text encapsulates information about common substrings in the text, we developed a technique based on the suffix tree data structure to estimate alphanumeric selectivity. We evaluate our methods empirically in the context of the TPC-D benchmark. We studied our methods experimentally against a variety of query patterns and identify five techniques that hold promise.

2.1.6 On-line algorithms and load balancing

In [8], we considered the load balancing problem, where there is a set of servers, and jobs arrive sequentially. Each job can be run on some subset of the servers, and must be assigned to one of them in an online fashion. Traditionally, the assignment of jobs to servers is measured by the norm; in other words, an assignment of jobs to servers is quantified by the maximum load assigned to any server. In this measure the performance of the greedy load balancing algorithm may be a logarithmic factor higher than the offline optimal. In many applications, the norm is not a suitable way to measure how well the jobs are balanced. If each job sees a delay that is proportional to the number of jobs on its server, then the average delay among all jobs is proportional to the sum of the squares of the numbers of jobs assigned to the servers. Minimizing the average delay is equivalent to minimizing the Euclidean (or L_2) norm. For any fixed p, $1 \leq p < \infty$, we show that the greedy algorithm performs within a constant factor of the offline optimal with respect to the L_p norm. The constant grows linearly with p, which is best possible, but does not depend on the number of servers and jobs.

In [9], we presented a natural online perfect matching problem motivated by problems in mobile computing. A total of n customers connect and disconnect sequentially, and each customer has an associated set of stations to which it may connect. Each station has a capacity limit. We allowed the network to preemptively switch a customer between allowed stations to make room for a new arrival. We wish to minimize the total number of switches required to provide service to every customer. Equivalently, we wish to maintain a perfect matching between customers and stations and minimize the lengths of the augmenting paths. We measured performance by the worst case ratio of the number of switches made to the minimum number required.
When each customer can be connected to at most two stations:

- Some intuitive algorithms have lower bounds of $\Omega(n)$ and $\Omega(n/\log n)$.
- When the station capacities are 1, there is an upper bound of $O(\sqrt{n})$.
- When customers do not disconnect and the station capacity is 1, we achieve a competitive ratio of $O(\log n)$.
- There is a lower bound of $\Omega(\sqrt{n})$ when the station capacities are 2.
- We presented optimal algorithms when the station capacity is arbitrary in special cases.

In [11], we first presented a simple algorithm for bin-packing that is worst-case optimal among bounded-space algorithms. However, it is not an online algorithm. We presented a new definition of lookahead for online algorithms, and show how to convert the simple algorithm into a more complicated optimal algorithm that is online with bounded lookahead. The main contribution of this paper may be the definition of online lookahead. Finally, we present experimental evidence showing that the basic approach works well on inputs drawn independently and uniformly from $[0, 1]$.

2.2 Approximate data structures

In [4], we introduced the notion of *approximate data structures*, in which a small amount of error is tolerated in the output. Approximate data structures trade error of approximation for faster operation, leading to theoretical and practical speedups for a wide variety of algorithms. We gave approximate variants of the van Emde Boas data structure, which support the same dynamic operations as the standard van Emde Boas data structure, except that answers to queries are approximate. The variants support all operations in constant time provided the error of approximation is

$1/\text{polylog}(n)$, and in $O(\log\log n)$ time provided the error is $1/\text{polynomial}(n)$, for n elements in the data structure.

We considered the tolerance of prototypical algorithms to approximate data structures. We studied in particular Prim's minimum spanning tree algorithm, Dijkstra's single-source shortest paths algorithm, and an on-line variant of Graham's convex hull algorithm. To obtain output which approximates the desired output with the error of approximation tending to zero, Prim's algorithm requires only linear time, Dijkstra's algorithm requires $O(m \log\log n)$ time, and the on-line variant of Graham's algorithm requires constant amortized time per operation.

2.2.1 Object complexity

In [10], we define a new complexity measure, called *object complexity*, for hidden-surface elimination algorithms. This model is more appropriate than the standard scene complexity measure used in computational geometry for predicting the performance of these algorithms on current graphics rendering systems.

We also presented an algorithm to determine the set of visible windows in 3-D scenes consisting of n isothetic windows. It takes time $O(n \log n)$, which is optimal. The algorithm solves in the object complexity model the same problem that Bern addressed for the standard scene complexity model.

2.3 Publications

1. P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter. "Indexing for Data Models with Constraints and Classes," submitted. A summary of earlier work appeared in *Proceedings of the 12th Annual ACM Symposium on Principles of Database Systems (PODS '93)*, Washington, D. C, May 1993.

2. M. H. Nodine and J. S. Vitter. "Optimal Deterministic Sorting on Parallel Disks," submitted. A summary of earlier work appeared in "Deterministic Distribution Sort in Shared and Distributed Memory Multiprocessors," *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '93)*, Velen, Germany, July 1993, 120–129.

3. M. H. Nodine and J. S. Vitter. "Optimal Deterministic Sorting in Parallel Memory Hierarchies," submitted. A summary of earlier work appeared in "Deterministic Distribution Sort in Shared and Distributed Memory Multiprocessors," *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '93)*, Velen, Germany, July 1993, 120–129.

4. Y. Matias, J. S. Vitter and W.-C. Ni, "Implementation of Approximate Data Structures," *Proceedings of the 5th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '94)*, Alexandria, VA, January 1994.

5. J. S. Vitter and E. A. M. Shriver. "Optimal Algorithms for Parallel Memory I: Two-Level Memories," double special issue on Large-Scale Memories in *Algorithmica*, **12**(2–3), 1994, 110–147.

6. J. S. Vitter and E. A. M. Shriver. "Optimal Algorithms for Parallel Memory II: Hierarchical Multilevel Memories," double special issue on Large-Scale Memories in *Algorithmica*, **12**(2–3), 1994, 148–169.

7. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. "External-Memory Graph Algorithms," *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, San Francisco, CA, January 1995.

8. A. Awerbuch, A. Azar, E. F. Grove, P. Krishnan, M. Y. Kao, and J. S. Vitter. "Load Balancing in the L_p Norm," Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS '95), Milwaukee, WI, October 1995.

9. Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. "Online Perfect Matching and Mobile Computing", *Proceedings of the Fourth Workshop on Algorithms and Data Structures (WADS '95)*, Kingston, Ontario, August 1995.

10. Edward F. Grove, T. M. Murali, and Jeffrey Scott Vitter. "The Object Complexity Model for Hidden-Surface Elimination", *Proceedings of the Seventh Canadian Conference on Computational Geometry (CCCG '95)*, Québec City, Québec, Canada, August 1995.

11. E. F. Grove. "Online Bin Packing with Lookahead", *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, San Francisco, CA, January 1995.

12. D. E. Vengroff and J. S. Vitter. "I/O-Efficient Scientific Computation using TPIE," *Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing (SPDP '95)*, San Antonio, TX, October 1995.

13. P. Krishnan, J. S. Vitter, and B. Iyer. "Estimating Alphanumeric Selectivity in the Presence of Wildcards," in preparation. A shortened version appears in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*, Montreal, Canada, May 1996, 282–293.

14. D. E. Vengroff and J. S. Vitter. "Three-Dimensional Range Searching in External Memory," *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, Philadelphia, PA, May 1996.

15. R. Barve, E. F. Grove, and J. S. Vitter. "Simple Randomized Mergesorting on Parallel Disks," to appear in a special issue on parallel I/O in *Parallel Computing*. An shortened version appeared in *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '96)*, Padua, Italy, June 1996, 109–118.

16. D. E. Vengroff and J. S. Vitter. "I/O-Efficient Scientific Computation using TPIE," *Proceedings of the Goddard Conference on Mass Storage Systems and Technologies*, College Park, MD, September 1996, published in NASA Conference Publication 3340, Volume II, 553–570.

17. L. Arge and J. S. Vitter. "Optimal Interval Management in External Memory," *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS '96)*, Burlington, VT, October 1996, 560–569. Also appeared in Abstracts of the First ARO CGC Workshop on Computational Geometry, Center for Geometric Computing, Johns Hopkins University, Baltimore, MD, October 1996.

18. G. Gibson, J. S. Vitter, and J. Wilkes, editors. "Strategic Directions in Storage I/O for Large-Scale Computing," *ACM Computing Surveys*, **28** (4), December 1996.

19. D. E. Vengroff. "A Transparent Parallel I/O Environment," *Proceedings of 1994 DAGS Symposium on Parallel Computation*, Hanover, NH, July 1994.

20. M. H. Nodine, M. T. Goodrich, and J. S. Vitter. "Blocking for External Graph Searching," *Algorithmica*, **16** (2), August 1996, 181-214. A shortened version appears in *Proceedings of the 12th Annual ACM Symposium on Principles of Database Systems (PODS '93)*, Washington, D. C, May 1993.

21. M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. "External-Memory Computational Geometry," *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS '93)*, Palo Alto, CA, November 1993.

22. L. Arge, D. E. Vengroff, and J. S. Vitter. "External-Memory Algorithms for Processing Line Segments in Geographic Information Systems," special issue on cartography and geographic information systems in *Algorithmica*, to appear. A shortened version appears in *Proceedings of the 3rd Annual European Symposium on Algorithms (ESA '95)*, September 1995, published in Lecture Notes in Computer Science, Springer-Verlag, Berlin, 295-310.

2.4 Participating scientific personnel

1. J. S. Vitter (professor)
2. E. F. Grove (assistant research professor)
3. P. Natsev (grad student—Earned M.S. degree while on project)
4. C. M. Procopiuc (grad student)

3 Report of inventions

The following patent has been issued to the PI under ARO support:

- Y. Matias, J. S. Vitter, and N. Young. "Method for Implementing Approximate Data Structures using Operations on Machine Words," United States Patent No. 5,519,840, Bell Laboratories, May 21, 1996.